

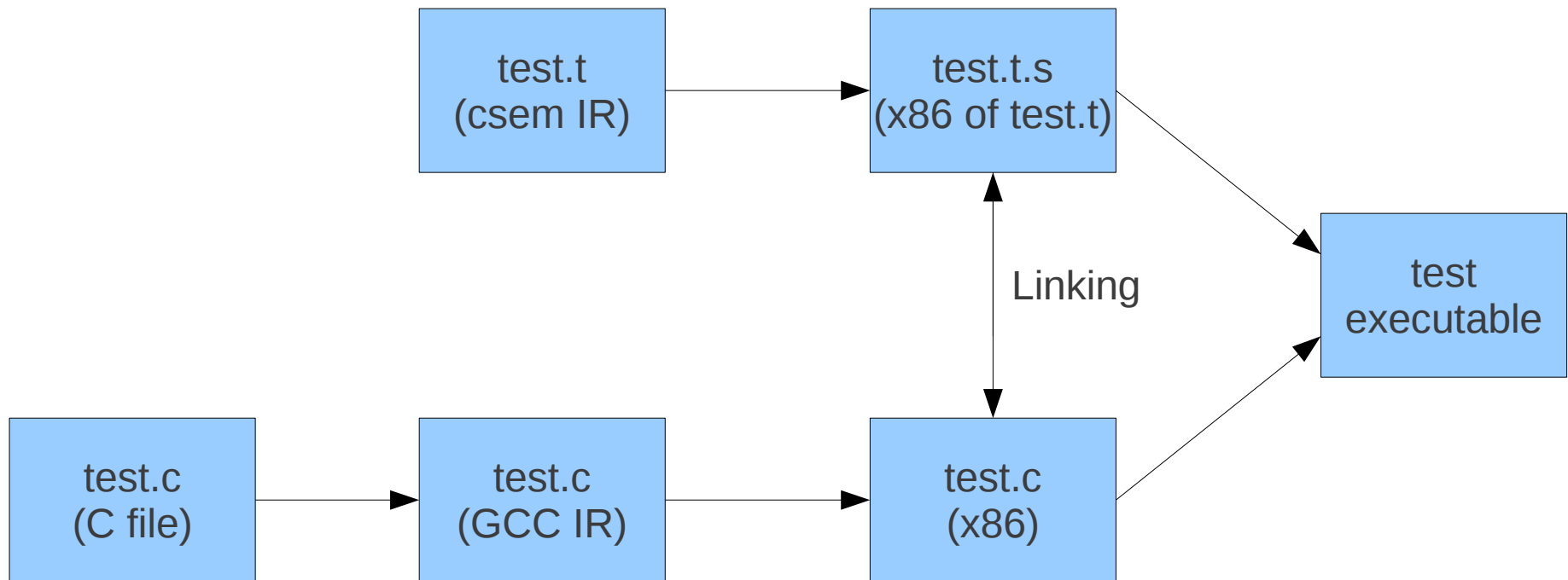
Code Generation for the x86 Architecture

EECS 665 Compiler Construction

Dr. Kulkarni

Marianne Jantz

Code Generation Framework for Lab 9



csem Intermediate Language Quadruples

- $x := y \text{ op } z$
 - Operate on y and z and place result in x
- $x := \text{global } name$
 - Yield address of global identifier $name$
- $x := \text{local } n$
 - Yield address of local identifier n
- $x := \text{param } n$
 - Yield address of parameter n
- $x := c$
 - Yield value of constant value c
- $x := s$
 - Yield address of character string s

csem Intermediate Language Quadruples (cont.)

- formal n
 - Allocate the formal having n bytes
- func $name$
 - Begin function $name$
- fend
 - End function
- bgnstmt n
 - Beginning of statement at line n

$name$ denotes an identifier from the C program. n denotes an integer. c denotes a C integer constant. s denotes a string enclosed by double quotes. x , y , and z denote quadruple temporaries. op denotes any of the C operators.

Assembly Generated for test.c by GCC Compiler

```
movl    $6, 4(%esp)
movl    $12, (%esp)
call    tstadd
movl    $.LC1, %edx
movl    %eax, 4(%esp)
movl    %edx, (%esp)
call    printf
```

Assembly generated by test.c's line 18:
`printf("12 + 6 = %d\n", tstadd(12,6));`

Assembly Generated for test.t by csem Intermediate Language Parser

```
.text
.globl tstadd
tstadd:
push    %ebp
movl    %esp, %ebp
subl    $256, %esp
movl    264(%esp), %eax
addl    268(%esp), %eax
movl    %eax, 272(%esp)
movl    272(%esp), %eax
addl    $256, %esp
pop     %ebp
ret
```

Assembly generated by test.t's tstadd()
definition:

```
func tstadd
formal 4
formal 4
bgnstmt 1
t1 := param 1
t2 := param 2
t3 := t1 +i t2
reti t3
fend
```

Assembly Generated for test.c by GCC Compiler

```
movl    $6, 4(%esp)
```

```
movl    $12, (%esp)
```

```
call    tstadd
```

```
movl    $.LC1, %edx
```

```
movl    %eax, 4(%esp)
```

```
movl    %edx, (%esp)
```

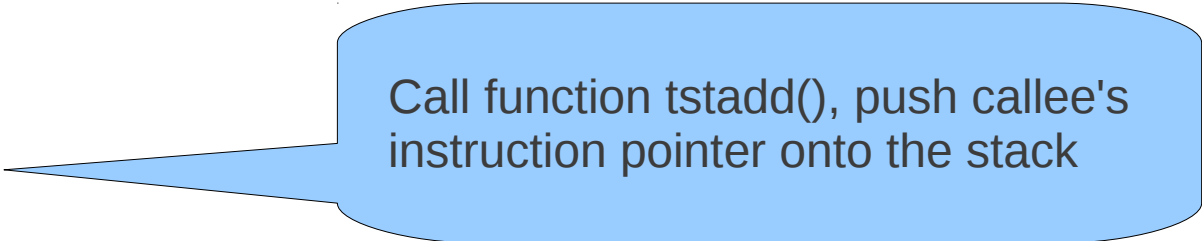
```
call    printf
```

Move arguments of `tstadd()` onto the stack to setup function call. Arguments are stored from right to left

The supporting routines of the provided source code will be able to use the stack offset to locate the arguments' position on the stack

Assembly Generated for test.c by GCC Compiler

```
movl    $6, 4(%esp)
movl    $12, (%esp)
call   tstadd
movl    $.LC1, %edx
movl    %eax, 4(%esp)
movl    %edx, (%esp)
call    printf
```



Call function `tstadd()`, push callee's instruction pointer onto the stack

Assembly Generated for test.t by csem Intermediate Language Parser

```
.text
.globl tstadd
tstadd:
    push    %ebp
    movl    %esp, %ebp
    subl    $256, %esp
    movl    264(%esp), %eax
    addl    268(%esp), %eax
    movl    %eax, 272(%esp)
    movl    272(%esp), %eax
    addl    $256, %esp
    pop     %ebp
    retthe
```

When the head of a function is parsed, a function header is printed in assembly

First, the previous frame pointer is saved (the callee's stack frame)

Next, the frame pointer is initialized to the bottom of stack, which grows downward. Now, we have a new local stack frame

Finally, we allocate space on the stack for the current routine

Assembly Generated for test.t by csem Intermediate Language Parser

```
.text
.globl tstadd
tstadd:
push    %ebp
movl    %esp, %ebp
subl    $256, %esp
movl    264(%esp), %eax
addl    268(%esp), %eax
movl    %eax, 272(%esp)
movl    272(%esp), %eax
addl    $256, %esp
pop     %ebp
retthe
```

Use the stack pointer to access arguments off the stack

Perform the addition operation on the parameters

The csem binary operation is parsed and this code is generated. The first operator is loaded from memory and placed in register EAX. Then, the second operator is loaded from memory and added to the value already in EAX

Assembly Generated for test.t by csem Intermediate Language Parser

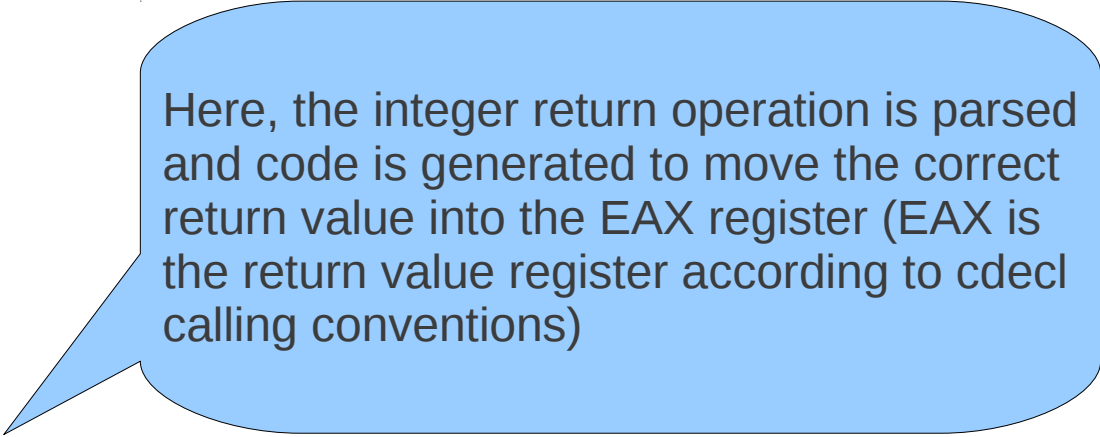
```
.text
.globl tstadd
tstadd:
push    %ebp
movl    %esp, %ebp
subl    $256, %esp
movl    264(%esp), %eax
addl    268(%esp), %eax
movl    %eax, 272(%esp)
movl    272(%esp), %eax
addl    $256, %esp
pop     %ebp
retthe
```

When parsing binary operations, the result of the operation is left in register EAX

When the assignment of a temporary is parsed, we can use this fact to make sure that we store the results in the correct local variable using a movl instruction and the stack offset of the routine

Assembly Generated for test.t by csem Intermediate Language Parser

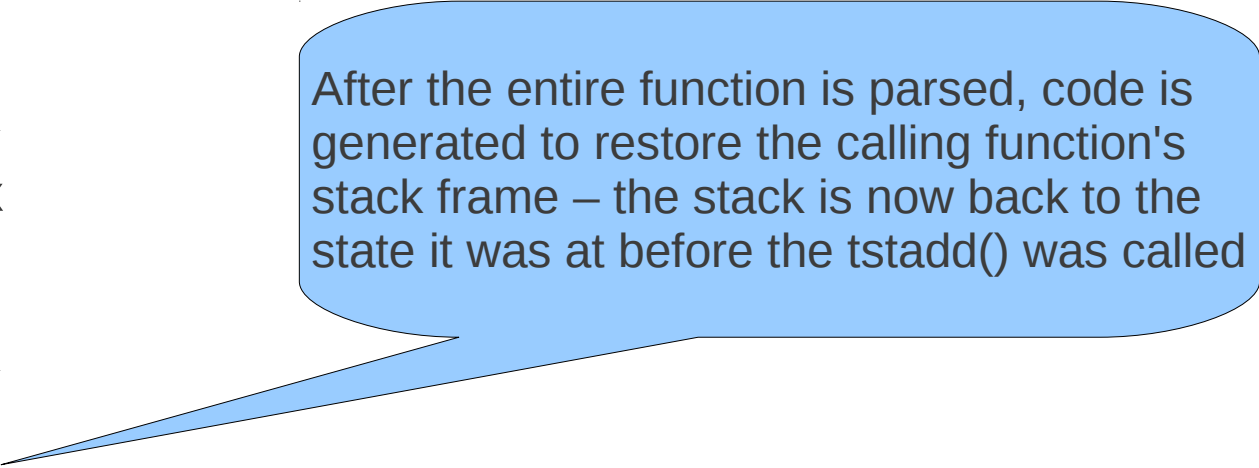
```
.text
.globl tstadd
tstadd:
push    %ebp
movl    %esp, %ebp
subl    $256, %esp
movl    264(%esp), %eax
addl    268(%esp), %eax
movl    %eax, 272(%esp)
movl    272(%esp), %eax
addl    $256, %esp
pop     %ebp
retthe
```



Here, the integer return operation is parsed and code is generated to move the correct return value into the EAX register (EAX is the return value register according to cdecl calling conventions)

Assembly Generated for test.t by csem Intermediate Language Parser

```
.text
.globl tstadd
tstadd:
push    %ebp
movl    %esp, %ebp
subl    $256, %esp
movl    264(%esp), %eax
addl    268(%esp), %eax
movl    %eax, 272(%esp)
movl    272(%esp), %eax
addl    $256, %esp
pop    %ebp
ret
```



After the entire function is parsed, code is generated to restore the calling function's stack frame – the stack is now back to the state it was at before the `tstadd()` was called

Assembly Generated for test.c by GCC Compiler

```
movl    $6, 4(%esp)
movl    $12, (%esp)
call    tstadd
movl    $.LC1, %edx
movl    %eax, 4(%esp)
movl    %edx, (%esp)
call    printf
```

The stack is setup to call the printf function, which will print LC1, a label of an ascii character string declared higher up in the assembly code, and the return value of tstadd()

Where to Turn for Help?

- Assembly References (listed on lab write-up)
- GDB
- -S flag
- ASK YOUR TA!